AUSTRALIAN UNIX USERS GROUP NEWSLETTER



## SUBSCRIPTIONS

As you all know, because you wont be reading this if you dont, all future issues of AUUGN must be payed for by the recipients. How many of the old readership have been conned into buying the second volume?

To date I have received about 27 subscriptions, and I think the flow has just peaked at a rate of about one every two or three days. The readership should level out at about 40. I have sent reminder notices, without AUUGNs, to the non-financial part of our old readership which may get a few more in.

One or two of you will find a note attached to this issue asking you to clear up some problems about payment of money. Please do this promptly as we cant carry much dead wood.

## GOODIES THIS ISSUE

Chris Rowles from Chem Eng at Sydney Uni tells us of his latest experiences with MINI-UNIX and PDP11/60s.

As a contrast to Chris's amusing style, John Lions has given me a series of technical papers, written while he was at Bell Labs. This issue I have included his gems called "Checking File Access Permissions in UNIX Systems" and "Shell Subprogram Facility", as well as some notes on security from Vrije. Next issue you may look forward to "Macros for Analyzing C Program Arguments". Makes your mouth water, does it not?

For those of you flogging dead PDP11/70s, a letter to John Lions from Purdue should be of interest. We certainly found it worth reading. Even though we had worked out most of what George Goble said, it was reassuring to know that other people had come to the same conclusions.

## OVERSEAS CORRESPONDENTS

Ian Johnstone has arrived safely at Bell, and says he will start writing as soon as the dust of his trip settles. Some of his rather hurried aerogrammes

to date give promise of some very interesting letters to come.

And "YES!" Ian, I will send you your copies of the Newsletter.

Our second correspondent will depart shortly for Antarctica. Eh? Where?

Dave Robinson (login name 'drob') has been posted, as part of his job, to the south pole. Dave is an avid UNIX user ('I had to use RSTS again today.... Yuk!') and fears loosing this mental prop so much that he is going to very great lengths to be able to 'dial up' from down south. We will assist him by tuning the radio to his frequency at set times, and if the ionosphere holds up, we should establish some sort of remote terminal record.

I dont know just what we will get out of the deal. The way DMR gets around the world Dave may come through with an interview. Still, there are always the penguins.....

---

VAX STUFF

---

There has been considerable comment about the "VAX-VMS wish list" included in the last AUUGN. Not all of it favourable.

Most people read it. A few understood it. A few wondered what it had to do with UNIX. The people who wrote it were not amused. DEC was saddened. I was quizzed.

Ianj is unrepentant.

Ian published it as a comparison between what some people wanted in VMS and what already existed in UNIX. I have offered DEC equal time (how many pages was it?) but as this issue spews from the copier I have not received a reply.

The VAX for Sydney Uni is in Australia. Where exactly in Australia is not known. For a while it seemed that H.M. Customs must be looking inside every chip in the machine, the length of time it took to 'clear customs'. Piers, Chrism, and the rest of the gang have a few bottles of champagne waiting. Odd that the bottles are not in the fridge.

---

NEWSLETTER CONTRIBUTIONS

---

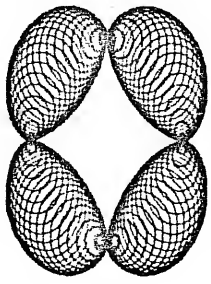
Come on all you people out there in Newsletter-land.

GET WRITING!!!!!!!!

---

***** SEASONS GREETINGS TO ALL OUR READERS *****

---



Peter Ivanov
Dept. of Computer Science
Electrical Engineering
PO Box 1
Kensington 2033
AUSTRALIA

(02) 662-3781

PROGRESS REPORT

MINI-UNIX/SU has undergone several improvements since it was released several months ago. These were documented as fix025 and fix026 in level 6 UNIX. The patches introduced the concept of INODE locking and fixed the sleep priority of unserviced buffer updates.

Further modifications were also made the UPDATE strategy. This involved removing the externally initiated update (/etc/update) and replacing this with a request internally scheduled. At present the update of the file system takes place once per minute (at 40 seconds past the minute), and updates only those buffers which have been modified since the last update and marked for write. Once the system has updated the buffer, the buffer has it's update request cleared. In MINI-UNIX this means that the system has one less process running (ie. /etc/update) and the system spends less time in updating buffers that have already been written to the system device. An undesirable by product of this strategy is that the super block is not updated as often and in the event of a system crash some timing information is lost.

One final improvement has been made to the update strategy to stop the updating of special files (or rather inodes associated with the special file eg /dev/tty8) as often as UNIX seems to desire. This prevents unnecessary i/o transfers to the system device updating the inode that was used to define the physical device of a special file. This was done because a problem appeared with floppy based MINI-UNIX. The inode associated with tty8 was updated every minute. This caused gross and unrecoverable physical damage to one block on the floppy (the oxide coating wears off the diskette).

These changes have allowed MINI-UNIX to be floppy based ("floppy UNIX"(?)). At present only two floppy drives are supported (AED 6200 and the AED 3100). This could easily be extended with a new driver for other types of floppy disks (eg. RX02 or if you really must the some-what slower and less desirable RX01).

Practice has shown that at least three floppy drives are the minimum desirable configuration for a self supporting MINI-UNIX system. One drive is filled up with system commands, compilers and Fortran plus swap area, another is useful as a system text file and the third is used as a user file system. The main limitation on the floppy based system is the small capacity of the floppy disks. To be a little critical, UNIX offers a wealth of useful processes, but they do occupy a lot of mass storage space. Two RK05 type disks help the system in terms of both speed (disk access) and storage capability (having most things on line), but floppy based systems are usable. One floppy based system has been running successfully for eight (8) weeks without any major problems.

The 11/03 systems have benefited from the the C-compiler written for the FIS option on the 11/40. This improves the execution speed of floating point software considerably (as the KEV-11 option is logically identical to the KE-11F on the 11/40).

DISTRIBUTION OF MINI-UNIX

Distribution of MINI-UNIX has been held up until it operated successfully for several weeks locally. This performance test has been achieved and the system is now in a reasonable state for distribution. Apologies are offered to the kind souls that have been waiting for a tape to

arrive, but the situation seems to be well in hand.

Next newsletter, I propose the include a list of parties interested in using MINI-UNIX, so those who would like to be included should drop me a line.

SOFTWARE ENHANCEMENTS (proposed)

RT-11 Fortran, Basic and MACRO are undergoing a facelift for MINI-UNIX. This is dependent upon re-writing the RT-11 LINKER to understanding the nature of the unmapped MINI-UNIX system. The work is nearing completion and the software will be available to those lucky users that can produce both a MINI-UNIX and an RT-11 licence.

A filter to extract floating point instructions and substitute them for a set of subroutine calls is also in hand. This is not quite ready yet. The idea behind this is to remove the need to emulate floating point instructions (and extended instructions) on 11/20 and such processors. This work is being done by John Holden at Sydney Uni.

I have heard some muttering of a network front-end based upon MINI-UNIX. At this stage they are only mutterings, but there is a good chance that something will come of this.

DISTRIBUTION PRE-REQUESTITES

1. A MINI-UNIX licence.

2. Provide one 2400 ft mag tape (or better still 3 x 600 ft mag tapes). The system comes as 3 x RK05 images on mag-tape.

3. As an alternative to 2. some bilateral communication may be necessary to provide a more convient distribution medium.

Chris Rowles
Department of Chemical Engineering
University of Sydney 2006.
(02) 692-2455

In September the Department of Chemical Engineering at the University of Sydney took delivery of a PDP-11/60. The system arrived with RSX-11M. After some experience with MINI-UNIX on our 11/20, it was decided to use UNIX as the operating system on the new 11/60. As UNIX was running at BASSER and at the other place (UNSW), we felt that 11/60 UNIX would be a relatively minor extension of an 11/40 UNIX system.

Our belief that conversion of UNIX to the 11/60 was a relatively simple matter was confirmed by the following quote : "We have a working UNIX system on an 11/60 in Western Australia " --- I.J.

How right he was. It even worked on RK06 disk packs. It worked on the eastern seaboard after a fashion, but that was RK07 based and somewhat of a new story. To be honest UNIX booted up from the RK07 packs and worked (as well as the original level 6 distribution could) within three working days of the 11/60 commissioning. It was then that the limitations of full UNIX became obvious. The solution appeared to move to UNSW UNIX and even BIGUNIX.

The solution was correct; but the path was somewhat long and poorly documented. This is an attempt to remember the exact trials and tribulations of a new UNIX coming out.

Technically all data contained in report is subject to change without notice.

## 11/60 --- WHAT IS AN 11/60 ?

The KD11K is a relatively new processor in the PDP-11 family. It lies somewhere between the 11/34 and the 11/70 in processing power. Technically it has an 11/40 type memory management unit, a maximum address space of 256K bytes, an integral floating point processor (implemented in the base machine's micro code) and a set of useful registers (accessable via the MED instruction). In performance, the 11/60 is between 0.4 to 0.8 11/70 performance. It's floating point unit is about 0.8 to 0.9 the speed of the 11/70. Integer instructions perform at about half 11/70 rate. The processor comes with a 2k byte cache, which gives the rather impressive performance.

The 11/60 comes with an expanded instruction set. Four new instructions were included to help with the debugging of the 11/60 micro-coded floating point instruction set. Two additional instructions are of general use to the UNIX programmer. XFC is an instruction that allows the user to access the writable control store option. MED is a new instruction that allows the programmer to access up to 92 internal registers. These are accessed via an escape sequence and the general register R0. The accessible registers are the general purpose register set, a spare 8 general purpose register set, the floating point registers and a set of error logging registers. The error log is generated as a CPU error occurs and allows error recovery via access to the internal processor state (jammed or held constant until released) via a set of six registers. This is a significant step forward in PDP-11 architecture.

The 11/60 also has a writable control store option, a hardware floating point unit and an 11/70 error register. The writable control store is accessible via a single instruction (again starting a multi-word escape sequence), XFC. Lastly, the 11/60 is complete as it has a STACK LIMIT register.

## FLOATING POINT --- MICRO-CODE STYLE.

The 11/60 has floating point instructions embedded into its micro-code. This is standard on all 11/60 processors. It also has an optional floating point processor (FP-11E). The conceptual problem created by the floating micro-code is "HOW does one support hardware that appears to be present, yet is not really there (or here or anywhere else !!) ?".

Expanding the conceptual problem a little, one sees that floating point instructions are emulated at the hardware level by the base 11/60. The micro-code behaves in a strange and poorly documented manner upon exception traps. Added to this feature, the floating point registers form the error log registers within the 11/60 CPU (unless the FP-11E is present). The support strategy seems to be to define the non-existant hardware and then blissfully pretend that it is really non-existant (or something like that).

The physical problem that arises from the floating point emulation is that floating point exceptions are treated as illegal instructions unless the FP-11E processor is present. This state of affairs was not immediately obvious and, needless to say not extremely well documented. The problem was overcome by writing a better back-off routine (based on the level 7 UNIX routines) and allowing the exceptions to bomb out in any manner they thought a reasonable idea. The 11/40 back-off tends to ignore floating point errors by saying that all floating point instructions are illegal. This does not allow the system to recover from an aborted instruction and prevents illegal instructions executing.

I feel that the full floating point unit will cause the processor to behave in a new and excitingly different manner, and it should be interesting to find out exactly what happens when it arrives next year.

It appears that it is possible to catch errors in an 11/70 manner using internally generated diagnostics however driving instructions (in terms of what the exact meaning of the error log registers) is held up by lack of DEC supplied printed words. Hopefully this condition will change with time as the 11/60 Technical Description Manual becomes available.

## THE ROAD TO BIG UNIX -- A GRAVEYARD OF 11/60 UNICES.

The road to BIG UNIX is rough (in the documentation sense). Once you have been there, all problems vanish ; getting there in the first place ---- well that is really this story.

There came to pass a state of UNIX that went BIG, and the people looked upon this and said it was good. Well some people looked upon it and said it was great, others said it was questionable and mapped buffers was a better path to follow. The difference of opinion appears to arise from the presence of an 11/70 memory management unit or the lack there of. Being of the second type I decided that BIGUNIX was not such a bad idea. The problem with BIG UNIX was that it was designed for the 11/40, and the 40 was the beginning and end of the processor types truely supporting BIG UNIX. With the advent of 11/34's some effort was made to support BIG UNIX.

If you ask a 34 owner what changes were made to support BIG UNIX you usually get the answer ....."well we had to make allowance for the lack of a STACK LIMIT REGISTER, and there was another mod, but I forget what it was... any way BIG UNIX works ". This state of affairs was somewhat confusing, so a fresh start was made upon the problem .... what is the essence of an 11/34 and what makes it different from an 11/40? This problem was expressed as

.... "what is the essence of an 11/60?" which seemed to be a better way of saying the same thing. The question was deemed to be answered when it was possible to define the CPUTYPE as 60 rather than 40 or 70. Clearly if this definition worked, then CPUTYPE definitions of 34, 23, 44, etc should follow in a relatively simple and logical manner.

I'M A 60 ---- WHAT MAKES ME UNIQUE ?

The answer is simple enough ---- you are a sixty (60) .. QED

UNSW modifications to the UNIX source had been defines in the DEFINES.H file. These patches to the original operating system were designed to differentiate between the PDP-11/40 and the PDP-11/70, and tend to lump all hardware features together.

Thus a host of mods that were deemed 11/40 or as 11/70 applied equally well to the 11/60. The initial approach to the problem was to say that 11/40 was almost 11/60, hence define 11/40 and patch anything else that was wrong. This approach fell apart when some of the 11/40 mods referred to the KT-11D memory management unit (and applied to the 11/60), while others referred to the absence of floating point instructions (a problem that arose above).

The solution taken was to redefine the patch levels to separate out the difference between members of the PDP-11 family into a more structured set of fundamental hardware differences. The first redefinition concerned the memory management unit type. The DEFINES.H file was altered to read MM-KT-40 and MM-KT-70 to differentiate between the memory management units (the two state KT-11D and the HEX state 11/70 type units). This meant that several 11/40 and 11/70 definitions were replaced and several of each type (non-memory management references still remained). The next step was to sort out the meaning of the remaining 11/40 references. These were references to the floating point unit and were redefined as such.

Hardware register differences were then defined in a semi-reasonable manner. For example a definition was made for a DISPLAY REGISTER, a SWITCH REGISTER, a STACK LIMIT REGISTER and a MEMORY PARITY REGISTER. The net result is to define the hardware in terms of a processor type number. This means that you define a processor type 11/60, 11/40, 11/70 11/34, 11/44 or 11/23 and that then defines the hardware registers and memory management unit type in use on the host processor. It also enables automatic selection of several UNIX operating system options designed to cope with hardware directly.

While this was not an earth shattering discovery, it should help UNIX to migrate to new processor types. It also removes the need to check CPUTYPE as an internal variable. This now means that this can be set to 60 or anything else you like, but 60 has special appeal.

But, still BIG UNIX failed to live on the 60 !

Defining PROCESS QUEUES, (a Chris Maltby patch and suggestion) solved the tendency for UNIX to die on the 11/60. To this day the insight that produced the results is a mystery. Even the great Maltby cannot explain why.

## I'M A 60 --- I THINK --- I RUN --- I STILL CAUSE HEARTACHE !

With BIG UNIX yielding to the 60, all problems appeared to be solved. It was then not unreasonable to try and bring all of the devices loaded on to the UNIBUS under UNIX. It was desirable to leave a MAG TAPE driver in the system, as this would allow BIG UNIX to run on the 11/34 at CSIRO. This would allow movement of source material to the 11/60 via the CSIRO's RK07 and a 9 track MAGTAPE.

It was also deemed essential to support our faithful PC-11 paper tape unit. The SFILE was created .... and the system was regenerated and then ..... it crashed upon booting. Well crash is an ugly word, died is a better description. The system would boot, access the disk in a feverish manner and die long before it tried to speak.

The PC-11 was removed and the problem disappeared.

Then the MAGTAPE was replaced by the PC-11 and the system remained alive and well.

Finally the MAGTAPE was returned ... instant death !

LESSON 1 : Standard UNIX loaders load up to 24k UNICES.... larger than this UNIX can only be loaded via a relocated loader and hence a self relocating BOOT.

The problem was that UNIX had grown beyond its ability to load itself. Fixing the loader allowed any logical device combination I desired and then some. In keeping with the true spirit of UNIX, a line between BASSER and CHEMENG is in the process of being laid. This will take the university some time to complete, but should be operational early 1980. In anticipation of the line being available, the multiplexed line driver has also been included into our system.

## RK07 BOOTSTRAPS --- AN ILLOGICAL SOFTWARE CONSTRUCTION

The RK07 provided untold experiences for both long established and novice UNIX gurus. To get an RK07 to run, a pack acknowledge command has to be issued to each drive before any data transfers can be made to the disk drives. Thus, the drives are provided with open and close functions. The close is necessary if the packs are to be swapped during the running life of a UNIX system. If this was not implemented then every time a new disk pack is loaded onto the system you need to reboot the system as well.

The bootstrap, however crashes if a pack acknowledge is attempted before the loader is relocated. Odd you may think, and with just reason. The accepted practice is not attempt to read from the disk without a pack acknowledge, which causes the read to fail. This invokes an error retry and a pack acknowledge. The problem then disappears and the system boots itself in a very acceptable manner. Intrinsically this is a very poor solution, but it works and I can not find the reason for a user initiated pack acknowledge to fail. Hence I have a very unacceptable (in a purist sense) bootstrap.

## I AM A 60 --- I THINK - I AM - I RUN -- I RUN RELIABLY AS BIGUNIX

The story ends.

The sequel begins. I have still to support the 11/60 in the nicest manner. One feature of the 60 that is desirable to support is its point of error logging capability. This means that at the onset of a cpu error, for any reason, the 11/60 stores the current internal state into 6 registers and then executes a CPU jam function. These registers can then be examined and the error condition can be corrected, without the need for the 11/40 type back off routines. Futhermore the error can be corrected at the site of occurrence and recovery is assured. This support is dependent upon the existance of the 11/60 CPU Technical Description.

The second feature to exploited within the 11/60 is its writable control store. This allows the 11/60 to extend its basic instruction set to include such little loverlies as :

| | |
|---|---|
| BLOCK MOVE ----- | 512 byte transfers form one instruction |
| CSAV | a useful thing to do from C. |
| CRET | the complement of CSAV |

The writable control store and the floating point hardware unit are items to be ordered in early 1980.

SYSTEM CONFIGURATION --- CHEMICAL ENGINEERING ; SYDNEY UNI.

CPU 11/60 with 256k bytes ECC MOS memory
Battery back up
2 off RK07 Disk Drives (28 Mbyte each)
Pertec DM3400 (5 Mbyte drive -- 2 x RK05 logically)
DU-11DA Synchronous Serial Interface (4800 bd to CYBER)
DZ-11A 8 line MUX.
2 off DL-11E Single line interfaces
DL-11W Single line interface
RT-11KRR Special line to 11/45 in Faculty of Engineering
Tally 2200 line printer
PC-11 Paper Tape Reader/Punch
10 off Terminals of various types
PDP-11/20 Satellite CPU.
PDP-11/03 Satellite CPU

Chris Rowles
Department of Chemical Engineering
University of Sydney.

# Checking File Access Permissions in UNIX Systems

*J. Lions*

Bell Laboratories
Murray Hill, New Jersey 07974

*ABSTRACT*

This memo discusses some proposed changes to the way file access permissions are checked in the UNIX† Time-sharing System.

## 1. Introduction

Many UNIX system installations are operated in a fairly free environment where the protection of files and the system from unwonted intrusion is not an issue. However there are more and more installations where protection is important and for which improvements to the present system are desirable.

This paper discusses some proposed changes to the way file access permissions are checked in UNIX systems. Its purpose is to provoke discussion of the issues before any changes are actually implemented.

## 2. The Present Situation

Each registered user of a UNIX system has a *user identification number (uid)*. There is one distinguished user, with a uid of zero, known as the *super-user*, who is able to bypass all the normal security barriers in the system. Each user is able to create and own files, and to set the access permissions for those files he owns. Usually there are a number of *pseudo-users* whose principal role is to "own" certain system-related files.

The UNIX system supports the concept of *user groups*, whose compositions are formally defined within the file */etc/group*. Each group has a unique *group identification number (gid)* and each user is a member of at least one such group. The file */etc/passwd* contains, for each user, the name of a group to which he or she belongs and this group is associated with the user when he or she logs in. A user may change his or her current group association by using the *newgrp* command, which creates a new shell with a revised group association.

Each process has associated with it two user identification numbers, which are commonly referred to as the *real uid* and the *effective uid*. For the majority of processes, these two values are identical and take the value of the user identification of the person who initiated the process. Also associated with each process are an *effective gid* and a *real gid*. When a process creates a child via the *fork* system call, the effective and real uids and gids are copied directly from the parent to the child.

For each file there are three sets of permissions which may be set by the owner of the file. The first set represents the privileges the owner allows himself, the second, the privileges he allows members of a group that he designates (not necessarily one of which he is a member), and the third, the privileges he allows to any user, who, being neither the owner nor a member of the designated group, can find the file by searching the directory hierarchy. (Within each set there are separate bits for read, write and execute permissions.)

For accessing a particular file, a process has the owner's permissions if the effective user is the owner of the file, or it has the group permissions, if the effective group is the same as the group associated with the file. Otherwise the process has only the permissions of the general user community. (It is somewhat paradoxical, but in many systems where groups are effectively ignored, so that all users belong to the same group, the *group* permissions are the

† UNIX is a trademark of Bell Laboratories.

ones which generally apply. Users tend to protect themselves against this knowledge by making the group and public permissions identical for their files.)

If the process executes (via the *exec* system call) a program file which has the *set user identification (setuid)* bit set, then at the time the program execution is initiated, the effective uid is set to the user identification of the person who *owns* the program (unless the effective user is already the super-user). (If the process *execs* a program which does not have the setuid bit set, then the effective and real uids and gids retain their prior values.) There is also a *set group identification (setgid)* bit whose effect is similar to that of the setuid bit. In general any process which is an execution of a setuid or setgid program will have a different file accessing capability from that which would hold otherwise.

The majority of *setuid* program files are owned by the super-user, who is granted permission to access any file. Normal users may invoke setuid programs to obtain services which the security barriers would otherwise prevent. For example, a normal user may not change the password file using the editor, but he may invoke the program */bin/passwd*, which is a setuid program owned by the super-user, to change that part of the file which contains his or her encrypted password.

Some manipulation of the effective uid is possible via the *setuid* system call. If the effective user is not the super-user, this call can only be used to change the identity of the effective uid to be the same as the real uid (i.e. to undo the effect of the setuid bit). If the effective user is the super-user, the call can be used to change both the effective and the real uids to any value. In both cases, a successful call on *setuid* leaves the effective and real uids with identical values.

The *access* system call has been added to the system to allow setuid programs (especially those owned by the super-user) to check access permissions based on the real uid. However it is not wise to expect that every setuid program will use this system call properly in every situation.

Since permission to access a file is determined only with respect to the effective uid and not the real id of the process, an attempt to reference a particular file may fail at present if the effective user is not the super-user even though the real user has permission to access the file. For example, *uucp*, which is a setuid program, owned by the administrative pseudo-user "uucp", is not able to copy a file, which does not have general read permission, even if the owner requests it. This is because the program has been made setuid in order that it may gain access to certain privileged files whose owner is "uucp" and which contain sensitive information about other systems.

One way to solve this problem is to make the super-user the owner of the program *uucp*. However this represents an overkill, since *uucp* would then have unfettered access to every file in the system, which it clearly does not need. Moreover it turns out every setuid file owned by the super-user is a potential "weak link" in the system security, and the number of such files should be kept as small as possible. (If a setuid file owned by the super-user is ever left in a writable state, an opportunity is generated for an unscrupulous user to overwrite it with another program, e.g. the shell, and hence gain super-user privileges.)

Another solution to the problem for *uucp* would be to use the existing facility for groups. It is a curious fact that in general there has been a reluctance on the part of the administrators of UNIX system installations to use groups in effective ways. The way it could be used in the above example is to invent a new group (call it "uucp" also) with a single member, the user "uucp". The privileged files owned by the user "uucp" would then be associated with the group "uucp", and allow access for members of that group. Programs owned by the user "uucp" that must have access his privileged files would then be made *setgid*, and would gain access via the group permission mechanism.

### 3. First Proposal

It is proposed that access permission to a file be granted to a process on the following basis:

1. If the effective user is the super-user, then the super-user's permission (universal access) will be granted; else
2. If either the effective user or the real user is the owner of the file, then the owner's permission will be granted; else
3. If either the effective group or the real group is the group of the owner of the file, then the owner's group's permission will be granted; else
4. The general user permission will be granted.

In brief, it is proposed to allow the process to combine the access capabilities of both the real and the effective user.* It may be pointed out that this does not create any new security hole in the system, since a setuid program already has the capability to make its effective user the same as its real user, and hence by a suitably devious arrangement of *fork* and *exec* can already combine the access permissions of real and effective user in any desired way.

### 4. Change to the Setuid System Call

This proposal is made with regard to the situation where a process executing a setuid program, creates a second process which in turn executes a setuid program which belongs to some third user. At present the real uid will be the same for both processes. Given the above change, the access permissions of the second process will be those of the real user and the second effective user (owner of the second program). (At present the access permissions are those of the second effective user only.)

An example where this situation is not entirely satisfactory occurs with the UNIX trouble reporting system. Any user can execute a reporting program, which is setuid because it accesses certain files that are protected. This program needs to shuffle some of these files around occasionally, but it cannot do so by forking and executing the *mv* command, because the latter is setuid and owned by the super-user. The *mv* command needs super-user privileges for the case where it is asked to move directories. If this is not the relevant case, it uses the *setuid* system call to change the effective uid to the real uid. There would be no problem if there were a way for the second process to have the combined file access permissions of the first and second effective users.

At present, if the effective user of the first process is not the super-user there is no way of achieving this effect. This will be possible if a (one-line) change is made to the *setuid* system call so that, in the case that the effective user is not the super-user, the effective uid and the real uid can be changed to the effective uid. In the example above, the *setuid* system call could then be used by the child process after the *fork* system call and before it performs an *exec* system call, to change the value of its real uid. A similar change should also be made to the *setgid* system call.

### 5. Alternative Proposals

There are two other proposals which have been suggested as alternatives to the above change to the *setuid* system call.

The first involves a change to the kernel *exec* procedure, which would require that, when a *setuid* program is being initiated, before the effective uid is reset, its value is to be copied into the real uid. This would solve the problem of the trouble reporting system, but may, as a side effect, affect some other as yet unrecognized system.

The second proposal, which has been implemented at the University of Waterloo, is to provide a system call (which they call "schizio"), which simply interchanges the effective and real uids of the calling process.

---

* The desired effect can be achieved via two simple changes to the kernel procedure *access*.

### 6. An Accounting Uid

If the above proposals are implemented, then use of the new *setuid* system call may erase the identity of the original user who initiated the process. It is important in some cases that this information not be lost. Accordingly it is proposed that a third uid -- the accounting uid -- be associated with each process. This would be set by an appropriate system call, which may be used successfully only when the effective user is the super-user, and which would be exercised in practice only by *login*. The accounting uid will be copied by *fork* and will ultimately be written into the accounting file by *exit*. Its existence will be transparent to the users in all other respects.

### 7. The Role of Groups

Since it has frequently been observed that many installations make no effective use of the *group* facilities, it may be timely to reconsider the role of these:

1. Are system administrators unable, or merely unwilling, to segregate their users into groups?
2. Is the proper of administration of groups simply too involved?
3. Would simple changes, such as to the *ls* program to print both uids and gids simultaneously, encourage the increased use of groups?
4. Should the concept of group be changed, for example so that groups become pseudo-users (with a unique user id), have entries in the password file, and are able to behave generally like a user? (Conversely, should a user be able to behave like a group, i.e. should each uid also define a gid?)

Perhaps the real question is whether there is a need for groups at all?

### 8. Concluding Remarks

It might be assumed that the proposals outlined above, by widening the powers of particular programs to access files, will in some way reduce the overall effectiveness of the file security mechanism. However there is a countervailing consideration, namely that the proposals will allow the access capabilities of programs to be tailored more closely to their needs. In particular, by reducing the number of setuid programs owned by the super-user, a major off-setting gain can be obtained.

# Shell Subprogram Facility

*J. Lions*

Bell Laboratories
Murray Hill, New Jersey 07974

*ABSTRACT*

The standard UNIX† shell program provides a subprogram facility, but expects subprograms to exist in the form of distinct files. This arrangement provides adequate functionality, but it is not always convenient from the programmer's point of view. A general shell command is described that allows a group of shell subprogram files to be packaged as a single file and invoked conveniently.

## 1. Introduction

The reasons for including subprogram facilities in programming languages are several and well-known. Thus it may come as a surprise initially that the language accepted by the UNIX shell includes no obvious subprogram syntax. Of course, the subprogram facility is not really missing, because one shell program may invoke another in a simple and straightforward manner. Moreover, parameters may be passed, there is no fundamental limit to the nesting depth of calls, and recursion is entirely possible. There is no disputing the generality of the subprogram facility provided. However because the shell expects separate subprograms to exist as separate files, and because a set of several small files is not always as convenient an object as a single larger file, it is possible to question the convenience of the existing arrangements.

Developing a moderately complex shell program (say of the order of 100 lines), or a group of shell programs, is often a qualitatively different experience from developing a program in C or in another high-level language:

1. One's initial knowledge of the tools being used may be incomplete, and the available documentation may need to be supplemented by experimentation.
2. Debugging shell programs has its own particular hazards, and it is desirable to develop and test large programs in several stages.
3. Detours may be needed to by-pass program bugs or features, or to supplement tools that are not quite sophisticated enough for the problem in hand.

For all the usual reasons, it is desirable to construct large shell programs as a series of smaller modules or subprograms, some of which may be extremely small (one or two lines). However since the present shell program expects these modules to exist as separate files, certain difficulties can arise:

1. Editing several programs simultaneously can be error-prone and is less convenient than editing one larger file. Moreover listing programs such as *pr* are not always convenient for processing lots of very small files.
2. The best way of dividing functions between subprograms may not be clear initially and may change during the development process. For example, a set of commands to print out the contents of a file may be located initially after the commands that created the file. Eventually it may be preferable to preserve those same commands elsewhere, in association with similar commands for displaying other files.
3. A group of commands may be factored out into a separate subprogram merely for the convenience of redirecting their output or input files.

† UNIX is a trademark of Bell Laboratories.

For these reasons the facility of packaging several subprograms into a single file is just as desirable for shell programs as for programs in other languages. The *spp* command described below provides this facility, using the facilities of the new Bourne shell. Its usefulness has been established, at least to the writer's satisfaction.

## 2. An Example

Let us use the term *package file* to denote a file that contains several shell programs or subprograms. An example of such a file is the following:

```
        cd ${1-.}
        for i in `ls`
        do
                tell $i
        done
tell:
        wc $1
        file $1
all:
        for i in `ls`
        do
                if test -d $i
                then
                        cd $i
                        echo $1/$i .....
                        all "$1/$i"
                        cd ..
                else
                        tell $i
                fi
        done
```

The structure of a package file is not unlike that of a *makefile*. It consists of a set of shell command lines, interspersed with *label lines*. Each label line consists of a label (currently restricted to be 1-14 lower case alphabetic characters) immediately followed by a colon. (Additional characters may appear on a label line after the colon, but are ignored.) Each label line marks the end of one subprogram and the beginning of the next, which it names. The label line

```
main:
```

is assumed to precede the first line of the file.

Assuming that the file above is named *show*, the command

```
spp show
```

has the following effect:

1. a temporary directory is created by *spp*, and three executable files, *main*, *tell* and *all* are written in it; the contents of the three files are lines one to five, lines seven and eight, and lines ten to twenty-one of *show* respectively;
2. the value of the environment variable $PATH is changed to include the temporary directory as the first entry in the command search path;
3. the command *main* is executed to begin the real work (that includes calls on *tell);*
4. the shell will find both *main* and *tell* (and *all* when needed) in the temporary directory;
5. finally, at the end of the computation, the contents of the temporary directory and the directory itself are removed.

Subprograms may be invoked individually. For example, if the command

```
spp show tell somefile
```

is executed, the same sequence of events as described above is executed except that the

command *tell somefile* is executed in place of the default command *main*. If it is desired to invoke the main program with parameters, the command must mention *main* explicitly, e.g.:

```
spp show main dir
```

would cause a different starting directory to be used.

### 3. The SSH Command

The shell program that implements *spp* can now be presented:

```
1  if /bin/test ! -r "$1"
2  then    echo Usage: $0 filename [ args ... ]
3          exit
4  fi
5  TMP=/usr/tmp/spp$$
6  trap "rm -f $TMP/*; rmdir $TMP" 0
7  trap "echo ; exit 1" 1 2 3 15
8  /bin/mkdir $TMP
9  /usr/lib/breakup $1 $TMP
10 PATH=$TMP:$PATH
11 export PATH TMP
12 shift
13 ${*-main}
```

The procedure checks that the first parameter is the name of a file that exists and can be read (lines 1-4); generates a temporary directory name (line 5); sets traps to ensure that the temporary files and directory will be removed (only once) when the procedure terminates or is interrupted (lines 6, 7); makes the temporary directory (line 8); executes the program *breakup* that copies the file that is its first parameter as a set of executable files (observing label lines) into the directory that is its second argument (line 9); changes the value of *PATH* (lines 10, 11); shifts the argument list to remove references to the first parameter (line 12); and executes the remaining parameter list as a command, or, if it is null, executes the command *main* (line 13).

The *breakup* program is given in Appendix A. Because both the shell and the editor lack good comment conventions, *breakup* introduces the following convention (which can be altered if necessary): each line is searched for the two character sequence "%%"; if it is found, it, all preceding white space and all succeeding characters are discarded before copying. Blank lines are discarded entirely.

Note that *ssp* exports the value of TMP, which is a shell variable naming the temporary directory. This allows small data files, for example, sets of commands for *awk* or *sed*, to be included usefully in the command file. (The path names for references to such files must begin with "$TMP".)

It is undesirable in general that subprograms be invoked asynchronously from within a package file because, before all references to them are complete, the temporary subprogram files may have been removed by *spp* as it terminates. There is no reason why the *spp* command itself may not be run asynchronously.

### 4. Another Example

Appendix B contains a non-trivial example of a package file that was developed during an investigation of the procedure calling relationships within the Portable C Compiler. The following comments may be of interest:

1. Most of the subprograms return their result via the standard output file. This is not essential but seems to be a good practice.
2. The file *temp* in the initial working directory is used as a temporary file, and is not explicitly removed.

3. *Awk* can be persuaded to use tabs as field separators, so some of the uses of *tr* are now seen not to be necessary. However *cref* deposits delete characters in its intermediate file after names that were eight or more characters, and *tr* seems to be the only way to remove these.

4. The code to remember the initial directory (as the variable *HERE*) seems sufficiently useful that it most probably should be included in *spp* itself.

5. *Cref* can become confused so that some ad hoc editorial changes are needed as part of the subprogram *crossref*.

6. There is no simple way of recognising the beginning and end of procedures in C program files. If programmers use a consistent style in laying out their programs, certain ad hoc editor scripts may just be sufficient.

APPENDIX A. breakup.c

```
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
/*      A program that reads the first named file, looking for "labels" and creates a
        file for each label from the statements that follow. The first file is called "main".
                A label is a line consisting of a name followed by a colon.
        Any characters after the colon are ignored. The name must consist
        of fourteen or less lower case alphabetic characters.
*/
char    line[260] = { ": main" };
FILE    *p, *f;
struct stat buf;
#define CD1 '%'
#define CD2 '%'

main (argc, argv)
int argc;
char *argv[];
{
        register char *q, *l, *k;
        if (argc < 3)
        {
nohope:
                printf("%s: args: filename directory_name\n", argv[0]);
                exit (2);
        }
        /* check for directory */
        if (stat (argv[2], &buf) || buf.st_mode & S_IFMT != S_IFDIR)
                goto nohope;
        if ((f = fopen (argv[1], "r")) == NULL)
        {       printf ("%s: can't open %s\n", argv[0], argv[1]);
                exit (2);
        }
        if (chdir (argv[2]))
        {       printf ("%s: can't change directories\n",argv[0]);
                exit (2);
        }

        l = &line[2];
        p = fopen (l, "w");     /* main */
        chmod (l, 0700);
        if (p == NULL)
        {
nocreat:
                printf ("%s: can't create %s\n", argv[0], l);
                exit (2);
        }
        while (fgets(l, 256, f) != NULL)
        {       /* check for 'label' line */
                for (q = l; q < &line[16]; q++)
                        if (*q < 'a' || *q > 'z')
                                break;
                if (*q != ':' || q == l)
                {       /* not a 'label' - look for comment */
                        for ( ; *q; q++)
                                if (*q == CD1 && *(q+1) == CD2)
                                {       while (isspace(*(q-1))) q--;
                                        *q++ = '\n';
```

```
                                        *q++ = '\0';
                                        break;
                                }
                        if (*l == '\n') continue;
                        fputs (l, p);
                        continue;
                }

                /* 'label' defines a new file */
                fclose (p);
                *q = '\0';
                if ((p = fopen (l, "w")) == NULL)
                        goto nocreat;
                chmod (l, 0700);
        }
        fclose (p);
        exit (0);
}
```

APPENDIX B. Sample Shell Package

```
set -x
FILES="/usr/src/cmd/pcc common trees.c local.c code.c pftn.c scan.c optim.c"
RECPROCS="where buildtre tsize fortarg clocal nncon talign ctype \
                optim offcon exname getlab"

        group list $FILES > plist; patchplist
        procs plist > procnames
        crossref procnames $FILES > ref
        order procnames ref $RECPROCS > porder
        printit

printit:

        pr acts1
        pr -e9 -w76 -2 -h "PROCEDURE LOCATIONS" plist
        printprocs procnames
        printcalls ref
        printorder
        : group "pr -x" $FILES

group:  GENERAL PROCEDURE TO OPERATE ON ALL FILES IN A GROUP
        set -x
        HERE=`pwd` ; export HERE
        job=$1 ; shift
        cd $1; shift
        for i in $*
        do
                $job $i
        done

list:   CREATE LIST OF: FILENAME, START, FINISH, PROCEDURENAME
        set -x
        lista $1 > $HERE/temp
        ed - $HERE/temp <<!
                1,\$s/^/ /
                g/(/.m.2
                g/(/.-2,j
                1,\$s/(//
                1,\$s/^/$1/
                w
                q
!
        cat $HERE/temp
lista:
        set -x

        ed - $1 <<\!
                g/^[a-z].*(.*)/s/(.*/(/p\
                .=\
                /^      }/=
                %% not everybody's convention
                Q
!

patchplist:     AD HOC CHANGES
        ed plist <<\!
                /\*relook/d
                w
                q
!
```

```
procs:  MAKE FILE OF PROCEDURE NAMES
        awk '{print $4}' $1

crossref:  LOOK FOR REFERENCES TO PROCEDURES

        set -x
        HERE=`pwd`
        procs=$1 ; shift
        cd $1 ; shift
        cref -cot $HERE/$procs $HERE/temp $*
        %% 'translate deletes left by cref to tabs '
        cd $HERE
        tr "\177\377" "\011\011"  < temp > tempb
        %% 'make file to list references to procedures '
        awk '{print $3, $1, $2, $4}' tempb ^ tr " " "      " > temp
        rm tempb
        %% 'kludge to overcome some idiosyncrasies .... '
        ed - temp <<\!
                g/RCON/s//fortarg/
                g/TNEXT/s//talloc/
                g/_/d
                w
                q
!
        sort -u temp

order:  TRY AND DETERMINE THE ORDER IN WHICH PROCEDURES SHOULD APPEAR
        awk '{print $1, $1}' $1 > temp
        awk '$1 != $2 {print $2, $1}' $2 ^ sort -u >> temp
        shift ; shift
        for i in $*
        do
                ed - temp <<!
                g/^$i/d
                w
                Q
!
        done
        tsort temp

printprocs:     DISPLAY PROCEDURES
        awk '{print $4, $1, $2}' $1 ^ tr " " "      " > temp
        pr -e9 -w78 -3 -h "PROCEDURE NAMES (in order of occurrence)" temp
        sort temp ^ \
          pr -e9 -w78 -3 -h "PROCEDURE DECLARATIONS (alphabetical)"

printcalls:     DISPLAY CALLER, CALLEE, LOCATION INFORMATION
        pr -e9 -w78 -2 -h "Procedure Calls, Ordered by Caller" $1
        sort +1 -2 +0 -1 +2 -3 +3n $1 ^ \
        pr -e9 -w78 -2 -h "Procedure Calls, Ordered by Callee"
        sort +2 -3 +3n +0 -2 $1 ^ \
        pr -e9 -w78 -2 -h "Procedure Calls, Ordered by File, Line Number"

printorder:     PRINT ORDER COMPARISON
        sdiff -w30 procnames porder ^
                pr -2 -w78 -h  "Actual v. Desirable Order"
```

# Vrije Security

Our system has been attacked by a number of very clever students in an attempt to crack our security. As a consequence of this we have discovered a number of security leaks in Unix, mostly due to the sloppy way of writing SUID programs. We have built a system to resist login simulators. The following bugs and changes have been made:

- Make login safer. Login simulators can be resisted by making a ".secure" file. affected files: login.c, su.c

- Core dumps from SUID programs were made with SUID permissions. Prior to making a core dump the permissions should be reset. Our students made links from /etc/passwd to core and wrote core dumps on top of the passwd file. This bug had already been fixed in version six and a half, but is present in some older versions. affected files: sig.c

- Make su safer. This is a well known bug. A user could open 15 files and then execute su. Su would fail to open the passwd file, assume there was no passwd file and make you super user by default. Su now checks for this contingency. affected files: su.c

- Make mv safer. There are two bugs in mv.

    1. When moving directories no checks were made on permissions at all. Think what would happen when you do this:
    echo "root::0:0::/:">/tmp/passwd
    chdir /
    mv etc etc2; mv tmp etc; su
    Mv now does lots of checking.

    2. Mv attempts to move things by linking and unlinking. Some students would start a number of moves in parallel, particularly moves of directories. Some of them would succeed in linking, but only one will succeed in unlinking, thus creating many links to a directory. Now mv unlinks the destination if the unlink of the source fails. affected files: mv.c

- Make the line printer daemon safer. The line printer daemon (lpd) would unlink any file if a -r flag was specified. A child can this is dangerous (opr -r /etc/passwd works like magic). Lpd now checks permissions. Lpd has to do this by finding out who executed the "opr". The only way is to check ownership of the dfa file and put put an unlink in command in it. The super-user's super-user). To check on this lpd demands the mode of a dfa to be 644 and the number of links to it to be exactly 1. affected files: opr.c, ppd.h, lpr.c

- Dissallow stty on someone else's terminal. Students used to stty to a different baud rate on super-used terminals to lure them away thinking their terminal broke down. Students also teased people they didn't like (assistants giving low grades) by setting their erase character to '\n'. This logged them off. affected files: tty.c, local.c

- Make mount safe. Mount checks the to-be-mounted file system for:

    1. SUID to another user

    2. SGID to another group

    3. Special files

4. link count mismatches of the dangerous kind (link>count)

5. dups

6. The usage count must be exactly one

Staff members can mount in spite of reasons 1, 2, and 3. affected files: pwd.c, sys3.c

- Make su safer still (Well known bug) If a passwd of exactly 100 characters is followed by its known encryption you will become the super-user. This was easily fixed. affected files: su.c

- The newgrp command is very dangerous. It needs a SUID but not SGID. If it has SGID it will always leave you staff. The fix is of course easy, but we dont need the command and have removed it. affected files: newgrp.c

- The passwd file had its temporary file in /tmp, a directory writable for anyone! Every user can remove the temporary file and replace it with one of his own choosing. Passwd will then overwrite the passwd file with the new temporary. We fixed this bug and introduced a new one! Within two days students discovered that no check was made on input length, so file descriptors were overwritten. Needless to say disaster loomed. This has also been fixed. affected files: passwd.c

- All default modes have been changed to 644 or 755. It is VU policy to have your files readable by anyone who cares to, but not writable. Many defaults used to be 666 or 777. affected files: MANY

- Change init to execute login in stead of a shell when the system is brought up single user. Now all students are allowed to bring up the system. The "rc" file contains a check on all file systems and will not mount bad file systems. This prevents the ruination of the system after a bad crash. affected files: init.c

- Chmod will not set the SGID bit if the file is owned by another group unless executed by the super-user. affected files: sys4.c

- Rmdir had to be rewritten. The old one would only unlink ".." when rmdir dir/../dir or similar was done. The new rmdir has been written in C, checks permissions and dose not contain this bug. Many students had secret sub-directories called "..", and so "find" would not find them. affected files: rmdir.c, rmdir.s

- Students can make programs that do chown on one of their files, call them "ls" or something like that and put them in likely places in the file system. When a super-user blunders past, and calls "ls" dreadfull things happen. To prevent this the search order of the shell has been changed: first /bin, then /usr/bin and finally the current directory. affected files: sh.c

PURDUE UNIVERSITY SCHOOL OF ELECTRICAL ENGINEERING

18 September, 1979.

Dr. John Lions
Computer Science Dept
The University of New South Wales
P.O. Box 1, Kensington,
New South Wales,
Australia, 2033

Dear Dr. Lions,

Our PDP-11/70 is also starting to bog down quite a bit now and there is not much more that can be done for it. We just got money approved to buy a VAX. Our job mix is fairly close to your job mix and we start bogging down at 55-65 logged in users although we have 112 "things" which can be logged into. It depends more on what users are doing than how many are on.

Summary of configuration for "A" machine 11/70:

PDP-11/70,1 Megabyte of core, FP-11C floating point
1 DEC RS04 disk on massbus
1 DEC RP04 disk on massbus
1 DEC TU16 magtape on massbus
1 SI (System Industries)/CDC 9766 300 MB disk on massbus
1 SI (System Industries)/CDC 9766 300 MB disk on Unibus
6 DH-11 (16 line serial multiplexors DMA output)

I would guess your system is bogging down from three things mainly. First your system has only 640K bytes of core. This will probably cause quite a bit of swapping with 40 users logged in.

Secondly, I assummed you have two disks on one controller, this generates a real bottle neck on I/O, especially if you have swaps going on also. I am not familar with the DM9100 drives, but other large drives like the DEC RP04, CDC9766, and CDC9762 (Dec RM03) run too fast for a Unibus interface. An 11/70 Unibus is a little slower than an 11/45 bus and many revolutions will be missed. 11/70 Unibus transfers also "cache wipe", they invalidate data in the CPU cache, slowing the CPU down. This is difficult to measure, but seems to be about 40-50% CPU slowdown when a continuous DMA transfer is in progress on the Unibus (Swapping) Massbus (cache bus) RH70 devices do not wipe the cache, plus they transfer 2 16 bit words at once so they can keep up with the super disks. On both of our 11/70's, each disk is on its own massbus controller. (one is on Unibus, but mostly for backup, etc) On the "a" machine, /tmp goes to first 1500 blocks of an RS04 (on dedicated massbus), primary swap goes to

Electrical Engineering Building
West Lafayette, Indiana 47907

last 547 blocks of RS04, secondary swap + the 2 most active user filesystems are on an RP04 (on its own massbus controller). Finally the root and 2 slightly less active filesystems are on a CDC9766 disk with an SI cachebus (massbus) controller. All user filesystems are 65500 blocks long, roots are 5000 blocks long, and total swap is about 7000 blocks. Root/swap areas (except RS04) are placed dead-center of the disks and an active filesystem goes on either side of it, to minimize seek time. Also our icheck tries to sort the freelist to interleave when possible on a track. Our disk drivers sort requests by cylinder. We have noticed that 3 massbus disks (each on own controller) seems to break the disk bottleneck on a large 11/70 system, and 1 megabyte of core seems to stop the memory bottleneck (swapping).

Thirdly, I believe DZ-11's interrupt on each character output. Running just 1 or 2 lines at 9600 baud doing continuous output will more or less consume an entire 11/70 doing interrupt processing/clist operations, causing output to come out in little spurts. We use DH-11's (and ACT DMAX-16's) which have DMA output. Our tty driver uses 512 byte disk buffers to queue up output for terminals over 2400 baud and this buffer is DMA'd out in one transfer. This also means you cannot have fill characters, etc on high speed lines. We have Lear Siegler ADM-3a CRT terminals which will run at 19,200 baud with no mods and at 38,400 baud with a minor clock/uart MOD and no fill chars required. About 3/4 of our terminals run faster than 4800 baud (9600, 19200 or 38400). We also have a 19200 baud serial line for RJE to a CDC 6500/6600 which runs more or less continuously during the days. An "ACT DMAX-16" is a DEC DH-11 equivalent, available from Able Computer Technology, 1751 Langley Ave, Irvine California 92714, USA. PH (714) 979-7030. A DMAX-16 looks like a DH-11 except it only takes 1/2 mounting space (2 hex cards), the parity control bit may be backwards (missprint in DEC manual), and speed EXTA is wired for 19,200 baud instead of crystal. We have had good luck with the DMAX-16, I believe a unit without modem control is $5,000 and with modem control is $5,900. Delivery is 2 or 3 weeks (in USA) verses over 1 year for a DEC DH-11.

We feel an "system" is more than just a CPU with the cheapest peripherals one can put on it. Much of our expansion money has gone into DH-11's (instead of DZ-11's), massbus disks instead of Unibus disks, etc. Since we do our own maintenance (DEC maint is horrible here!) we can choose "brand X" devices, etc. Many of these are bad bets though, have heard of many problems with DATARAM memory, DIVA and AMPEX (and DEC) and Pertec (Computer LABS) disks have been known to have been to cause many problems. We have had good luck with SI (System Industries)/CDC disks and memory from "Standard". We had a few problems with the early SI 9400 Unibus controllers, but they bent over backwards and got us going. We spent quite a bit of time tracking down Unibus protocol problems in their early 9400 controllers last fall, and they gave us 2 free massbus adaptors ($5,000 Ea) for our effort. Their current version of both the Unibus and massbus controllers seems solid and handles power fails correctly. For more info, you can call SI (Sunnyvale, Calif) at 408-732-1650.

There have been quite a few software changes made to our system (plus home brew network by Bill Croft) over the last 4 years or so. A copy of the EE system is available free to anyone

with a Version 6, PWB or Version 7 UNIX license, just send us a 2400' tape and a copy of your license, with return postage. The Toronto Users' group meeting software is also on the tape. Did you get the Purdue tape from the June 1979 Toronto conference? We have kind of frozen our V6 system now and are beginning the painful conversion to Version 7. Version 7 seems to have a lot of neat "ivory-tower" features to keep the UNIX hackers happy, but it is far from running like greased lightning to support the 70 to 80 logged in users we will have to run by next semester. Below is a piece of the talk I gave to the Toronto users group meeting and Bell Labs which is sort of a summary of major changes at Purdue/EE:

## Purdue/EE Kernel

* Power fail recovery - save and restore regs

* 1400 user pop, 63-64 Max users login, NPROC 250, 16 bit uid, no gid

* 96 DH-11 Ports + 16 Network "pseudo ttys"

* Extended Addressing for Disk buffer pool and other tables
  Allows upto 130 buffers.

* Inode cache - inactive incore inodes booted out on LRU basis.

* TTY driver (on DH-11) does DMA output from DISK buffers
  which eliminates clist slowdown on 4800 baud to 38,400 baud.

* High speed (upto 19,200 baud) serial input (on DH-11) in
  8-bit binary (upload) with no protocol.

* Split swap - Primary (90% swaps) go to 548 blocks of RS04 Fixed
  head disk. Overflow goes to RP04 secondary swap (10000 blocks).
  Procs inactive for 30 sec are migrated from primary to secondary swap.

* Several misc sched changes: After 30 sec CPU time, proc is considered
  to be "background" and will only get swapped in when "easy" core
  is available. Interactive running proc won't be booted out to make
  room.

* nicer sys call - does a nice to another running proc.
  # nicer -127 pid  causes pid to not get scheduled (stepped)
  # nicer --5 pid (or better) gives extra favors to proc

* Most kernel printf's go to clist (except system disk errors)
  where they are read from /dev/errlog by user proc.

* stty spy (write only bit) causes ttwrite() and ttyinput() to
  put copy of chars on /dev/statlog clist (like /dev/errlog)
  for monitoring other terminals.

* force tty input. If su and seek(fd, -1, 0), then output sent
  to fd (a /dev/tty?) is written to input clist instead. When
  done to a raw disk file, this invokes boot from disk.

* Timelimit sys call - sends signal when USER+SYS exceeds limit
  (works like alarm call)

Most of our load breaks down as follows: About 25% are students running DEC's F4P (fortran-IV-Plus), CULC has converted to run under UNIX. About 25% C programs, 25% cross assemblies for 8080 and 6800 microprocessors and about 25% nroff. Almost all of the users logged in are doing something. About 40-50 of the terminals are more or less "public" and are available to most EE students round the clock or 18 hours/day. There are about 50 more terminals in Prof's offices, labs, secretary typing pool, etc and most of these are on use 8-5. We are going with the administrative UNIX license (being worked out with Western now). Many of the professor ports are shared between several offices. We have a terminal switcher (128 ports cpu+tty) about 90% built, but the person building it, left for a better job.

As for performance, it takes 1-2 minutes to Fortran compile or CC compile a typical 100-200 line program during the afternoons when load is the wost. We typically max out between (55 and 65) users logged in, 350 incore inodes, 350 NPROC, NBUF 120. We run 2 electrostatic (Houston Inst) 2400 LPM lineprinters, and a Centronics on DH-11 serial lines. Almost all terminals are Lear ADM-3a's and run at high baud rates (mostly 9600 or 19200) baud, many users use the "ned" (Rand Editor) two dimensional editor. We have also rewritten Bell's ed and added many features.

As for increasing performance, there is nothing left, all the blood has been squeezed out of the rock. Next step is a new machine (VAX) running UNIX which may be here in April. We will have to run 80-85 logged on users in the next few months, but are going to use the network to fork off CC compiles, F4P compiles, and nroff's to other network machines and move the necessary files around.

For your machine, I would guess you have been getting very good performance considering the peripherals you have (DZ-11's and Unibus disks). Even with souped up peripherals, you could only go to about 60 users before going totally CPU bound. It is probably time for another machine. Bell Labs gets another CPU for every 20 logged in users or so. We are not quite that rich and have to stretch a litter farther.

Feel free to call or write me if you have any more questions which I may be able to help you with. Although our distributions are not as nice and elegant as Berkeley's, you are welcome to them.

Sincerely,

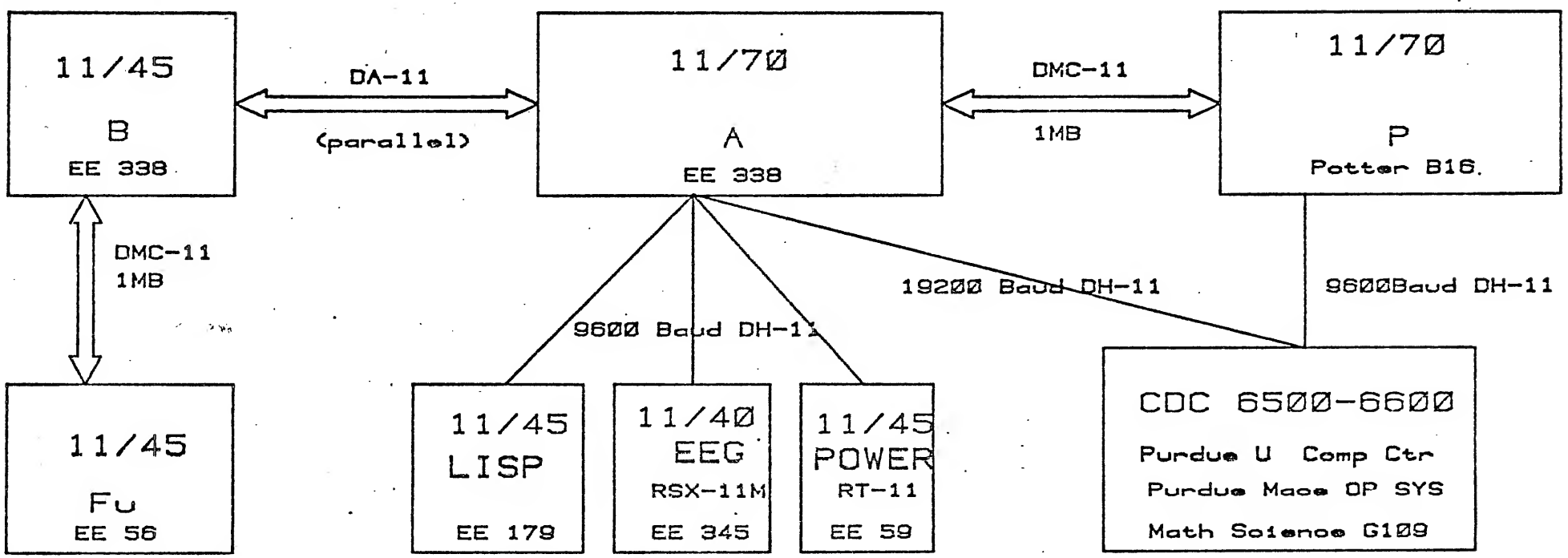
George H. Goble

George H. Goble

School of Electrical Eng.
Purdue University
West Lafayette, Indiana 47907

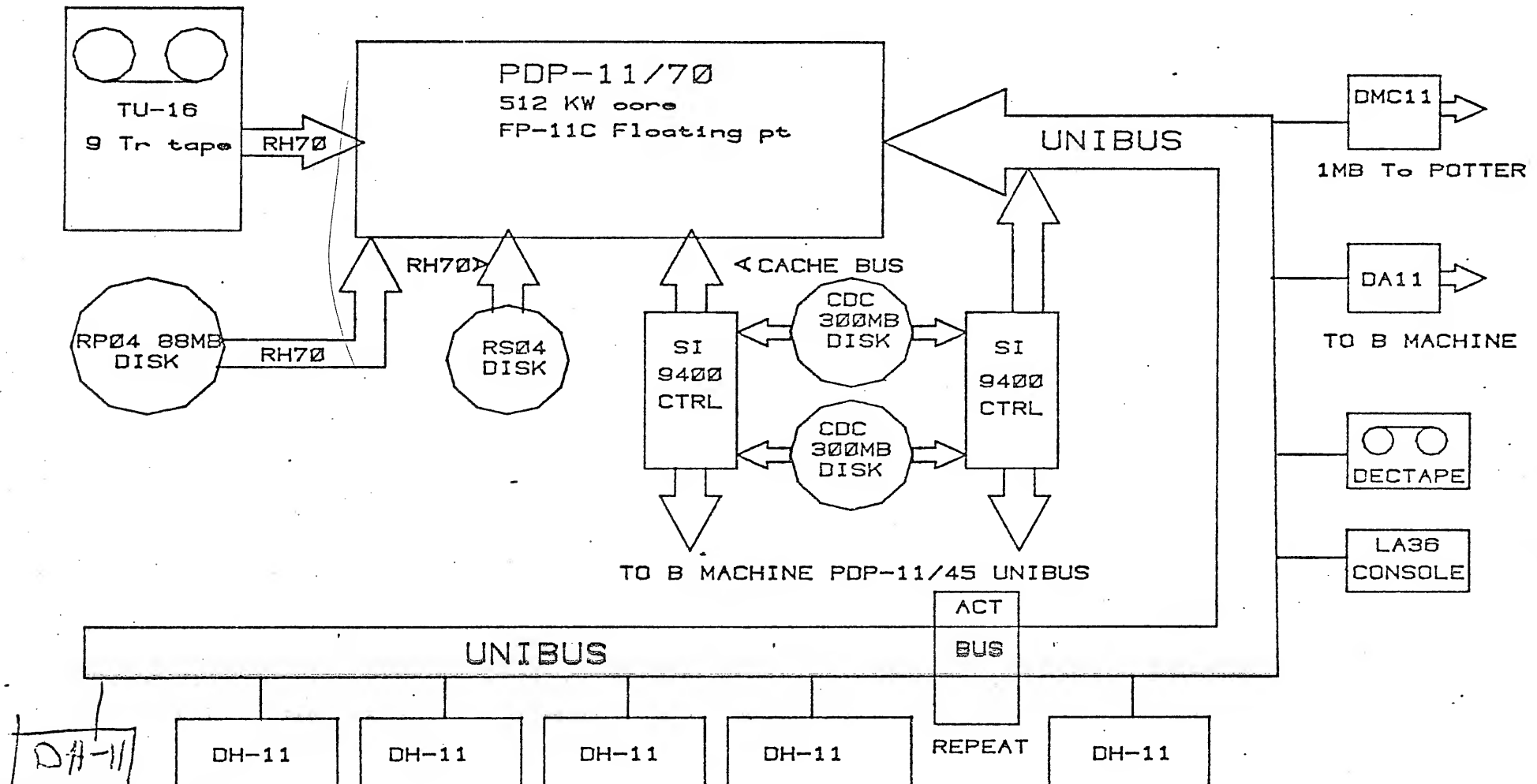
Phone: (317)-493-3890

# Purdue Electrical Engineering Computer Network




All operating systems are UNIX
unless otherwise specified

# "A" machine configuration



# Victoria University of Wellington

*Telephone 721-000*

*Private Bag*
*Wellington*
*New Zealand*

Department of Information Science

26 October 1979

Mr. C.D. Rowles,
Dept. of Chemical Engineering,
University of Sydney,
N.S.W. 2006,
Australia.

Dear Mr. Rowles,

We have just acquired a Mini-Unix licence and plan to install it on a PDP-11/10 with 28k words and two RL01 disk drives. It will typically be used by one or two users in a laboratory environment for our advanced OS course. There are several ways that you could be of help to us:

i) Could you please supply us with a copy of your reliable version of Mini-Unix? A tape is enclosed and a copy of our licence attached. We will be happy to pay any service fee that may be necessary.

ii) Do you know anyone running Mini-Unix with RL01 drives. The University of Canterbury is running UNIX on an 11/34 with RL01's. From your experience could we expect much trouble adding their driver to Mini-Unix?

Thank you in advance for any help you can offer.

Yours sincerely,

John H. Hine
Senior Lecturer in Computer Science

Encl.

Hebrew University
Computer Science Lab
Manchester House
Jerusalem
20/11/79

Peter Ianov
Dept. of Computer Science
Univ. of New South Wales
P.O. Box 1
Kensington 2033
Australia

Dear Peter,

I wish to thank you for sending us your newsletter. We recieved a VAX recently, and are in the process of installing it. Our configuration is almost non-existent: 256K bytes of memory, a single RM03 disc drive, a DZ11 mux and a TE16 magtape. On the way are 512K bytes and a DMC11.

Naturally, the VMS-or-UNIX discussion is going on here, too (quite ferociously). The fact that the VAX/UNIX isn't here yet weakens our (the UNIXers) position, yet some SCOPE-like features of VAX/VMS scare the local CDC refugees off. I keep feeling dizzy whenever I see a full VMS file name.

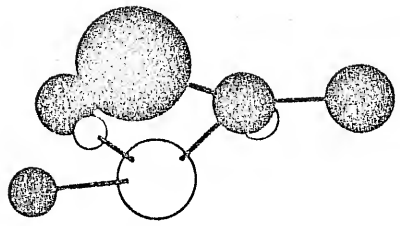
We have lately written for our 11/45 an IBM-format diskette handling program, and just finished implementing a loadable system-calls and drivers facility (it also enables writing, utilizing and dropping them while the system is running). The latter will soon be ready for distribution.

About the "wish-list" you mentioned - has anyone written a decent CRT scroller for UNIX, that includes ed's pattern search capability?

We would appreciate any advice about using the VAX, especially about reliability and security of the two available systems.

Yours sincerely
Gabi Steinberg
Gabi Steinberg

P.S. We have started with the bureocratic procedure to obtain the $22 for the subscription + back issues.



**QUEENSLAND INSTITUTE OF TECHNOLOGY**
GEORGE STREET, BRISBANE. TELEPHONE 221 2411.
P.O. BOX 246, NORTH QUAY, QUEENSLAND, AUSTRALIA, 4000.

29th October, 1979.

Mr. C.D. Rowles,
Dept. Chemical Engineering,
University of Sydney,
N.S.W. 2006.

Dear Mr. Rowles,

This department has recently purchased an LSI 11/03 with 56 Kb of memory, dual RXV21-BD floppy disc drives and 4 line serial interface. I am interested in the prospect of implementing mini-UNIX which would presumably need to be done from magnetic tape on the DEC-10 system, which is shortly to be delivered to the Institute, down the communications interface. The department does not at this stage own a mini-UNIX licence.

I would be grateful for your comments on the above and for details of the cost of purchase of UNIX and a current mini-UNIX licence.

Yours sincerely,

(DR) GEORGE MOHAY.
Senior Lecturer,
Dept. Mathematics and Computer Science.

IN REPLY PLEASE QUOTE
REF. IF:MJC

TELEPHONE: ARMIDALE 72 2911
AREA CODE 067
TELEX NUMBER 66050
POST CODE 2351

# THE UNIVERSITY OF NEW ENGLAND

ARMIDALE, N.S.W.

DEPARTMENT OF COMPUTING SCIENCE
30th November, 1979

Mr. P. Ivanov,
Editor,
AUUGN,
The University of New South Wales,
P.O. Box 1,
KENSINGTON. N.S.W. 2033.

Dear Mr. Ivanov,

Mitchell Duncan has passed on to me your letter from 10th October in which you give the information about Unix.

I am in charge of UNIX at U.N.E., and I was not aware of the Australian Unix Users Group. This is perhaps the reason why you do not have any record about us.

We do have a license (I am enclosing a photocopy of the relevant pages of it), and we would appreciate it if you would inform us about any Users Group Meetings.

I am also sending a subscription for the Newsletter, but this has to go through our library.

At the present there is just one thing which is important to us and we would welcome your help with it. Our terminals on the PDP 11/34 are on DZ-11 multiplexors (or most of them are), and there is no DZ-driver on Unix. I wonder whether you happened to have such a driver, and if so, whether we could get it from you. This would save us writing it which is a bit difficult with our current knowledge of UNIX.

I hope you will be satisfied that we are legal users of UNIX - so that you can now discuss matters with us.

Looking forward to hearing from you,

Yours sincerely,

Ivan Fris.

We have a number of DZ drivers in use locally. Some flash-fast (Ianjs), some a little slower. We cant send you any software unless we know how to send it. Do you have a mag tape? Kevin Hill from the AGSM will contact you soon to clear up this point, and arrange a distribution of locally modified software.
I have credited you with a newsletter subscription. Please ask the library to send MONEY, not order forms and other paper warfare.....

Peteri

6/11/79.


# L. E. R. S.

## Laboratoires d'Études et de Recherches Synthelabo

*58, rue de la Glacière - 75621 PARIS Cedex 13*

Société Anonyme au Capital de 2.700.000 F. R. C. Paris B 69 1294 Téléphone : 589-89-29


October 30, 1979

Peter Ivanov
U.N.S.W.
Australia

Dear Dr Ivanov,

I was very interested to receive your latest Australian Unix Newsletter. It is very good that you and the UK Unix User Group manage to produce such good newsletters even if the US one has died. We would like to subscribe airmail to your newsletter and receive backnumbers airmail; please let us know if you require extra money for airmail, meanwhile we will send the ordinary subscription.

Last Friday myself and a colleague went to the UK Unix User Group meeting at Newcastle, which although we missed half of the meeting was very interesting. We are trying to get a European Unix User Group set up perhaps as a SIG of DECUS Europe. There is already a Dutch Unix User Group.

Currently myself and the head of our statistics and pharmacokinetics group are looking for personnel. I enclose a copy of the advertisement which I circulated at Newcastle. The Statistics group job is less UNIX orientated but needs someone with statistics, pharmacokinetics and database experience or interest. I must add that in general there are problems employing Australians in France (work permits) but if we are presented with really good candidates we are willing to try.

I hope to be in Australia in October 1980 for IFIP 80, would it be possible to arrange visits to UNSW or an Australian Unix Group meeting around that time ?

Yours sincerely,

Ian R. Perry

Ian R. Perry

Groupe Informatique,
L.E.R.S. - Synthelabo,
58, rue de la Glaciere,
75013 Paris,
France.

29th November 1979

Dear Dr Johnstone

At present my company is trying to replace two computer scientists. One of the two computer scientists that we take on will be very much involved with using our version of unix, the other may sometimes be using unix. I hope that as a fellow unix user you will be able to help me by placing the second page of this communication on a suitable notice board in your department.

Yours sincerely

Ian R. Perry

(Ian R. Perry Chef du Groupe Informatique)

A major French pharmaceutical company requires for its research centre in the Paris area:

TWO COMPUTER SCIENTISTS

with the following minimum qualifications:

1) a degree in computer science or an equivalent qualification

2) experience of programming in FORTRAN and at least one of the following languages: ALGOL, APL, BASIC, C, PL1 or PASCAL

3) experience of signal processing or EEG analysis or biomedical research or statistics or IMAGE or other database management system

Experience in some of the following would also be desirable:

a) Tektronix 4051
b) numerical analysis
c) Hewlett Packard 1000 computers
d) electronics
e) DEC PDP11 or LSI11 computers
f) the UNIX operating system

Ability to speak or read French is not essential, however, a working knowledge of English would be useful.

If interested send curriculum vitae to:

Dr I.R. Perry,
L.E.R.S. - Synthelabo,
58, rue de la Glaciere,
75013 Paris,
France

or telephone Paris 589 89 29 extension 230

Department of Computer Science

THE UNIVERSITY OF AUCKLAND

PRIVATE BAG AUCKLAND NEW ZEALAND TELEPHONE 792-300

4 December 1979

Dr Peter Ivanov,
Department of Computer Science,
University of New South Wales,
P.O. Box 1,
Kensington 2033,
AUSTRALIA.

Dear Dr Ivanov,

A contact at the University of Canterbury, New Zealand, tells me that your Department is the headquarters for the Australian UNIX Users' Group and that you have recently set up a UNIX newsletter, AUGGN. I have been highly interested in UNIX for several years now, but until recently have had neither the equipment nor the justification for it. Now at last we are setting up a Computer Science Department at the University of Auckland and we are hoping to get a PDP-11 on which we can run UNIX. We are still at the stage of applying for grants, however, and even if everything progresses smoothly we would not have it before September 1980 at the earliest. Which brings me to my first query: is it possible for us to subscribe to AUGGN, and order all back issues, without yet having a UNIX licence? If so, we would be most grateful if you could set the necessary wheels moving; I understand that the subscription is $12 per annum, and the back issues $10 the lot. If this proves possible, the rest of my questions may well be answered by the newsletters; please don't waste your time answering them individually if this is the case.

I understand that version 6 UNIX has been superseded by version 7, and that the new version requires a PDP 11/44 or upwards. We were intending, however, to order only a small LSI 11/23 System, reputed to be software-compatible with the PDP 11/34, so presumably we would be forced to stick with version 6. I would therefore be interested to know if Algol-68 (available for $250 from an English University, I'm told) and Fortran-77 could be made to run under version 6.

Finally, I understand that MINIUNIX is operational in the Chemical Engineering Department at your university; perhaps someone there would be able to answer my remaining questions. We have several PDP 11/20s and a PDP 11/10 around the university and some of the owning departments might be interested in MINIUNIX. I understand that it will support three or four users (presumably on a 28k word system) but I am interested to know how much memory the resident operating system requires, i.e. how much is available for each user. Also, I would like to know what problem-oriented languages are available; perhaps you have an overview note that

Continued...

you could pass on? On the assumption that MINIUNIX is in fact available for distribution, what medium/media do you distribute it on? Obviously we would need to obtain a licence from Bell Labs first; would a Version 6 licence suffice or is a special licence needed?

Well, that's the end of my questions; I hope they do not waste too much of your time. Many thanks for any help you can give.

Yours sincerely,

R. J. Lobb

Richard Lobb